PIMS

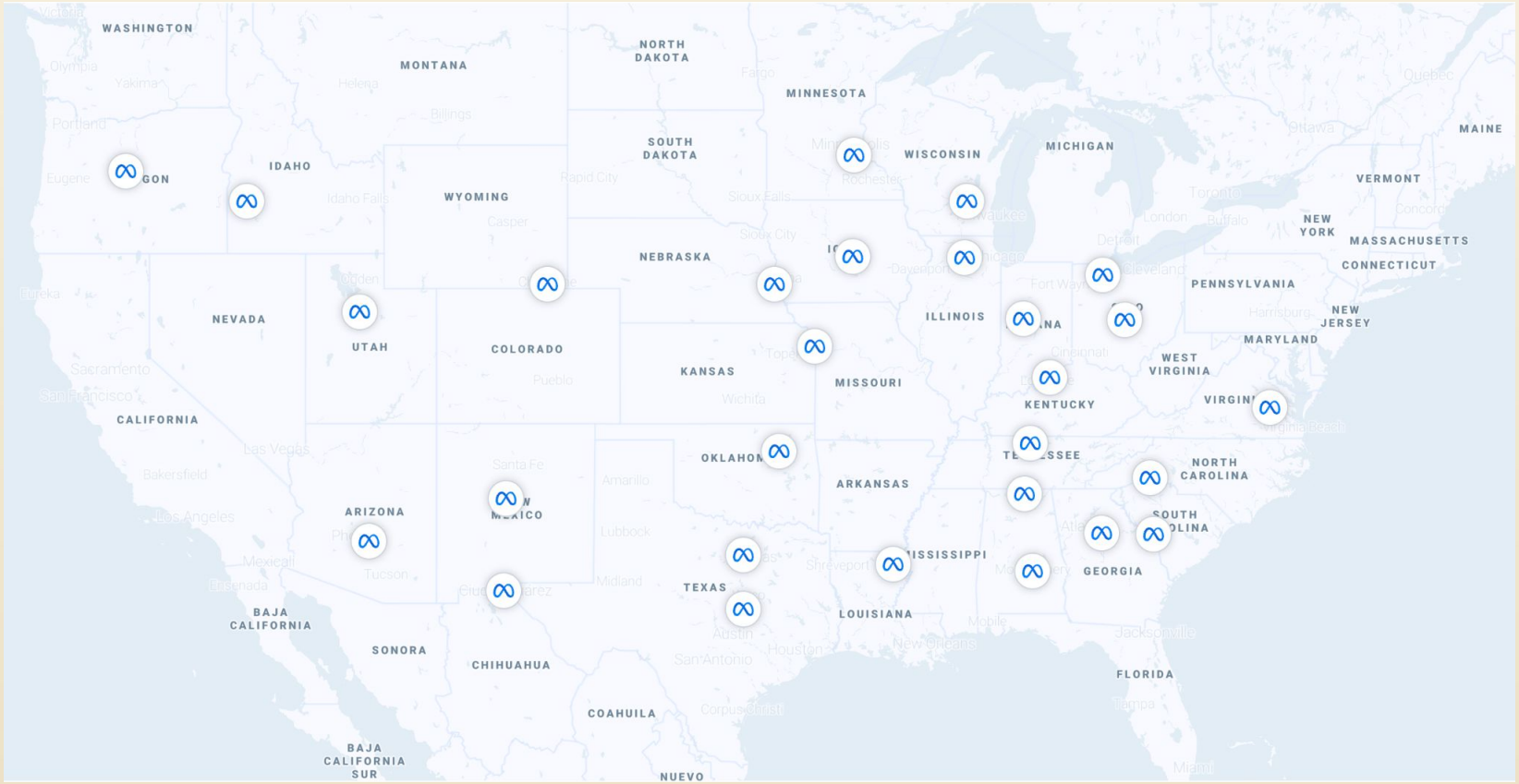
Fleet-wide Datacenter Maintenance at Meta

Benjamin Leonhardi et al
Speaker · Amitav Mohanty

Agenda

1. **Context**
   - 1.1 Cloud → regions
   - 1.2 Cloud → machines and services
   - 1.3 Maintenance types
2. **The IAAS contract**
3. **Why is the contract necessary?**
4. **How is the contract achieved?**
5. **What the IAAS contract gives us**

Datacenters



Context

The cloud, at scale



10M+

servers



10k+

services

Millions of machines running thousands of services — one fleet to maintain

Maintenances

	Physical	Software
Planned	- Physical large scope - Physical rack level	- Software upgrades
Unplanned	- Rack-level repairs - Asset-level repairs - Large-scope	- Leaks - Regressions

Today we cover planned maintenances only

Goals for planned maintenance

Two kinds of work — two promises

Planned · Physical



Seamless support

swaps, repairs, expansion —
with no service disruption

Planned · Software



Predictability

upgrades reach the fleet on a
known, bounded schedule

Both delivered within the IaaS contract

Agenda

1. Context
2. The IAAS contract
   - 2.1 What is a domain?
     - 2.1.1 Types of domains
   - 2.2 Domain alignment
   - 2.3 Region → domains
   - 2.4 The contract
3. Why is the contract necessary?
4. How is the contract achieved?
5. What the IAAS contract gives us

What is a domain?

A domain is a unit that **fails together**

Carved by two criteria

Physical utility

shared physical resource

Logical function

shared logical role

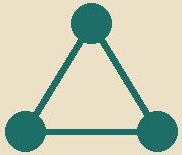
Domain types

Example criteria for grouping



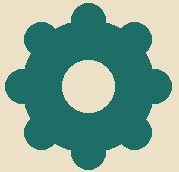
Power

shared power feed



Network

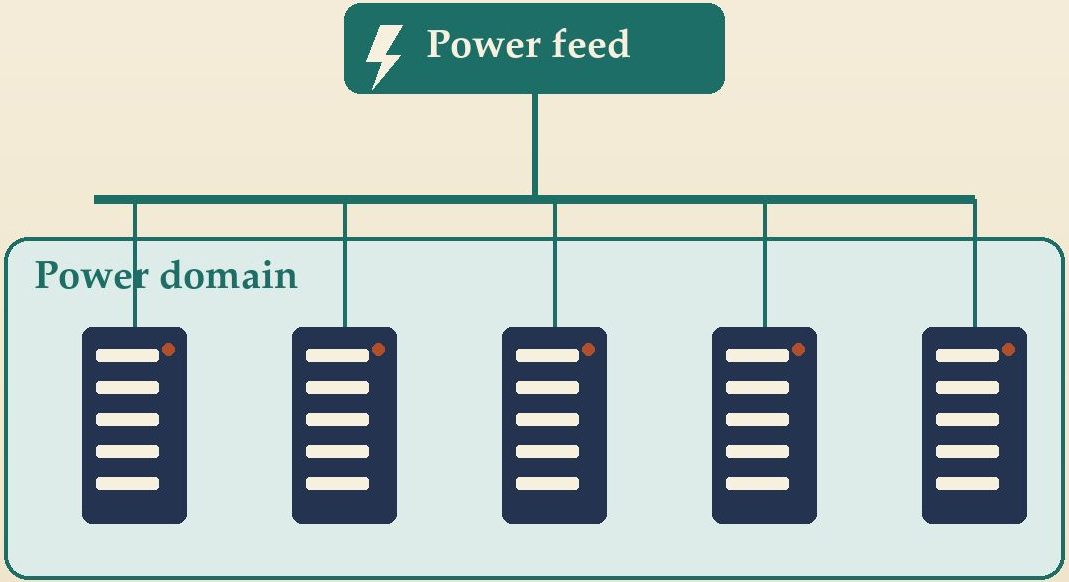
shared network fabric



Maintenance

worked on together

A power domain

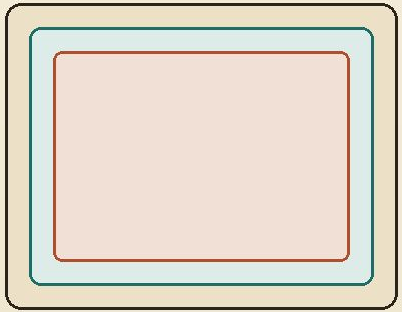


Machines on one power feed fail together

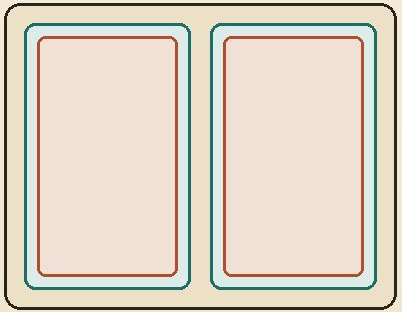
Base tenet: aligned domains

Power and network domains nest inside the maintenance domain

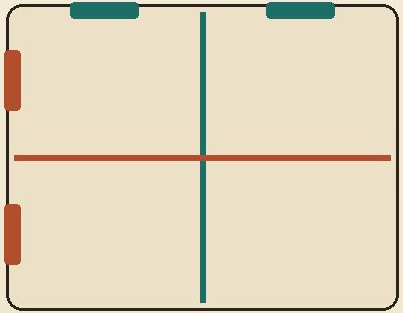
Power Network Maintenance



power = network
= maintenance



power = network
×2 per maintenance

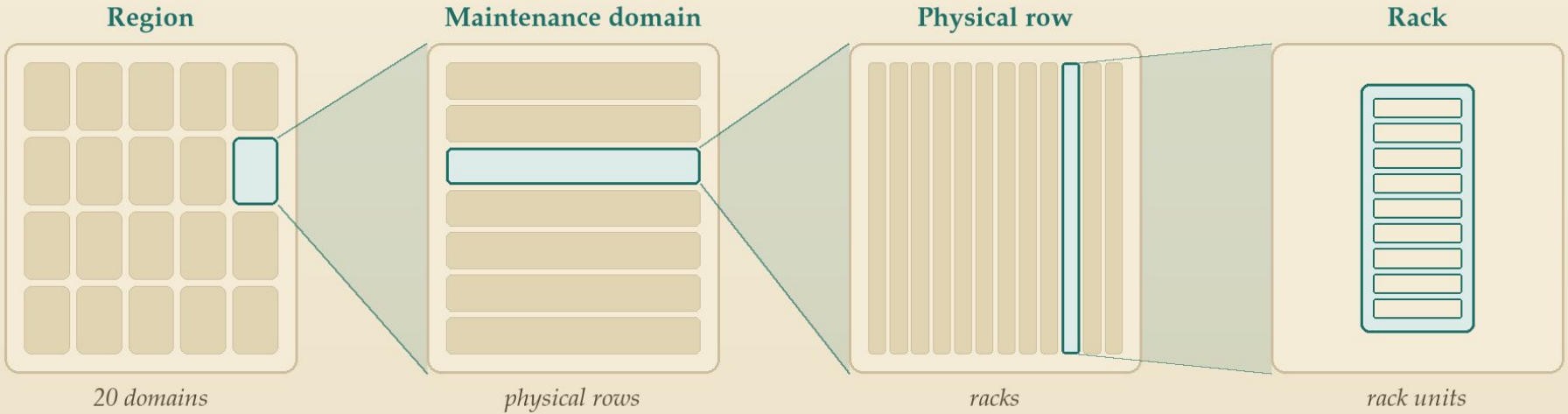


power × network
2 × 2 grid

Examples only — any such nesting works (N power = 1 maintenance, same for network)

Maintenance domains

How a region looks, zoomed in

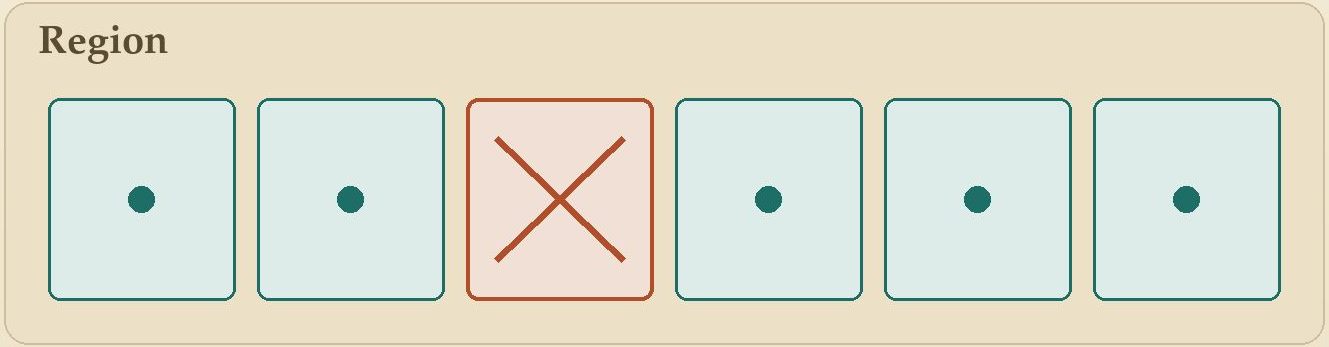


A region zooms from 20 domains down to rows and racks

The IAAS contract

At most **one domain** is down at a time

per region · for planned operations



Every other domain keeps serving

The IAAS Contract

WHY

Physical and automated maintenances

HOW

Even spread of h/w & s/w



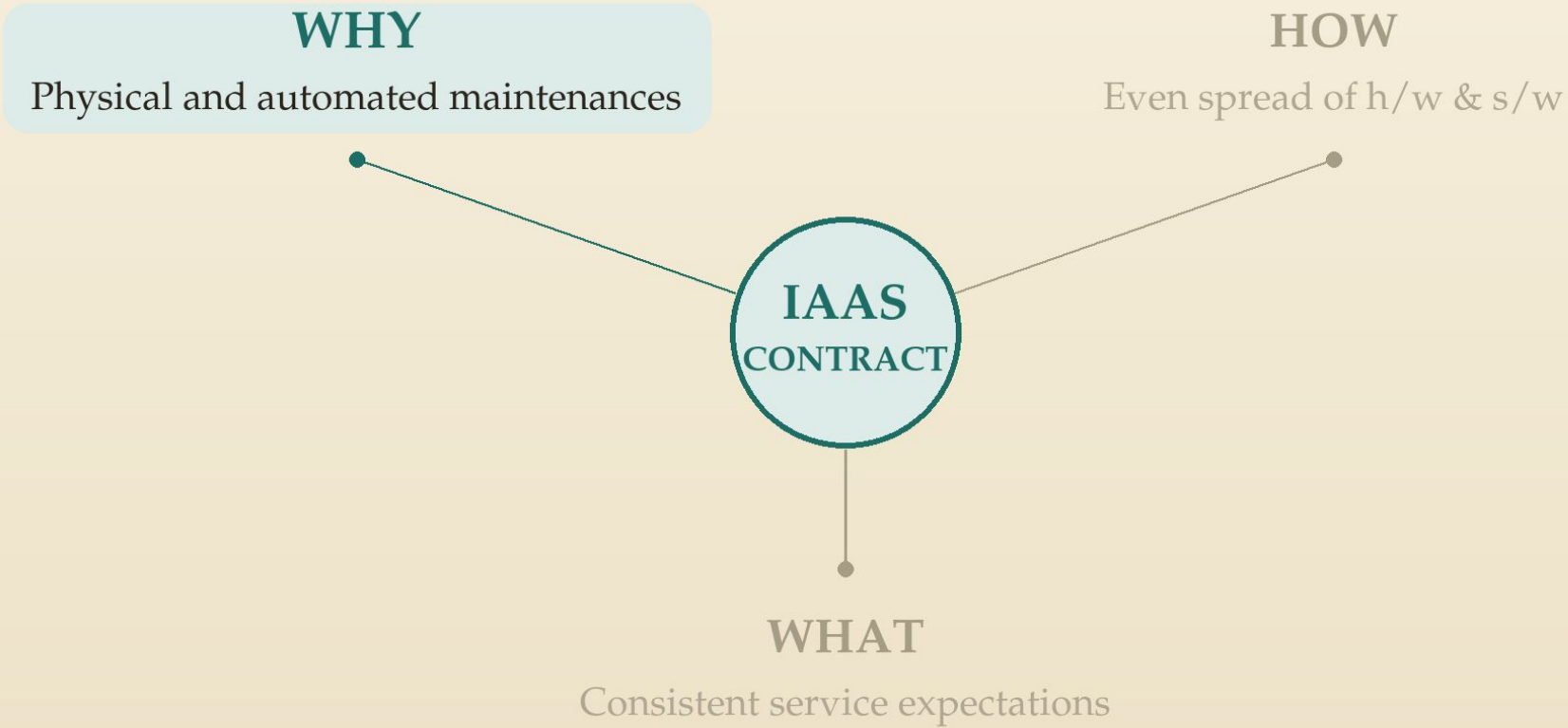
WHAT

Consistent service expectations

Agenda

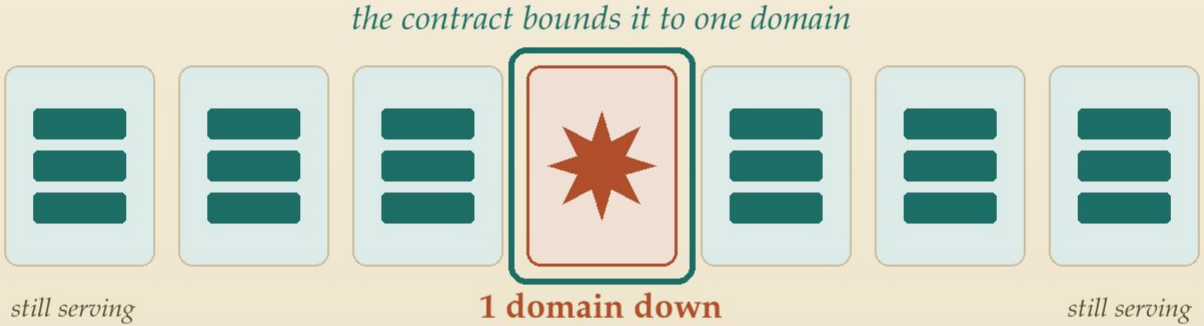
1. Context
2. The IAAS contract
3. **Why is the contract necessary?**
   - 3.1 Absorbing loss of a domain
4. How is the contract achieved?
5. What the IAAS contract gives us

The IAAS Contract



Why the IAAS contract?

To bound the impact of any maintenance



A domain down stays a domain-sized problem

the impact is contained — never region-wide

One domain down — is the impact even?

each colour is a different service

1 MD ≠ 1 service domain



One domain down guts one service

1 MD = 1 service domain



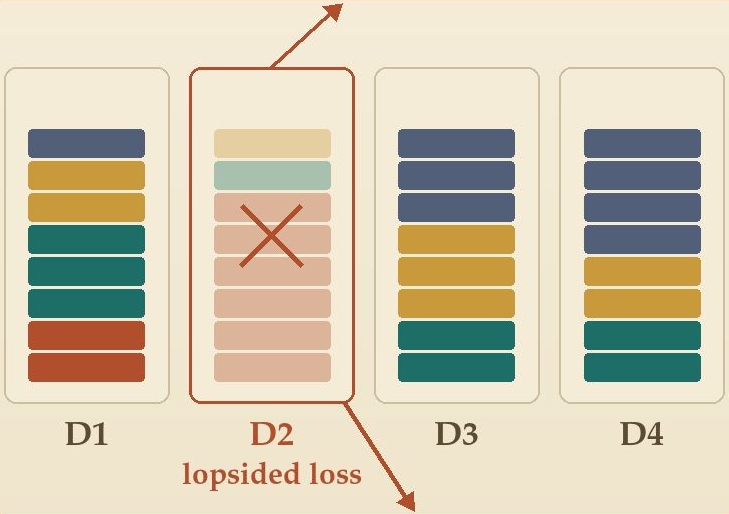
Every service loses the same small slice

Even spread turns a domain down into a small, expected loss

Who cares if a domain is down?

Without contract

Service owners — ✗ Unhappy

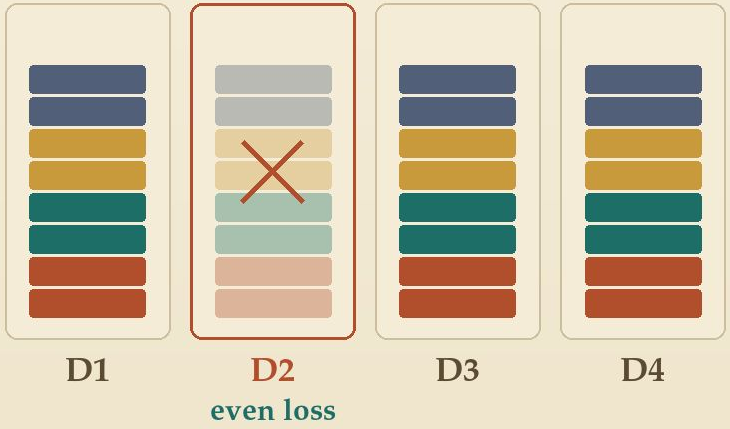


Ops — ✗ Unhappy

With contract

Service owners — ✓ Happy

No coordination needed



Ops — ✓ Happy

Equivalent domains make one domain down a non-event

Agenda

1. Context
2. The IAAS contract
3. Why is the contract necessary?
4. How is the contract achieved?
   - 4.1 Spread s/w and h/w
   - 4.2 Multiplexed buffer for 1 domain
5. What the IAAS contract gives us

The IAAS Contract

WHY

Physical and automated maintenances

HOW

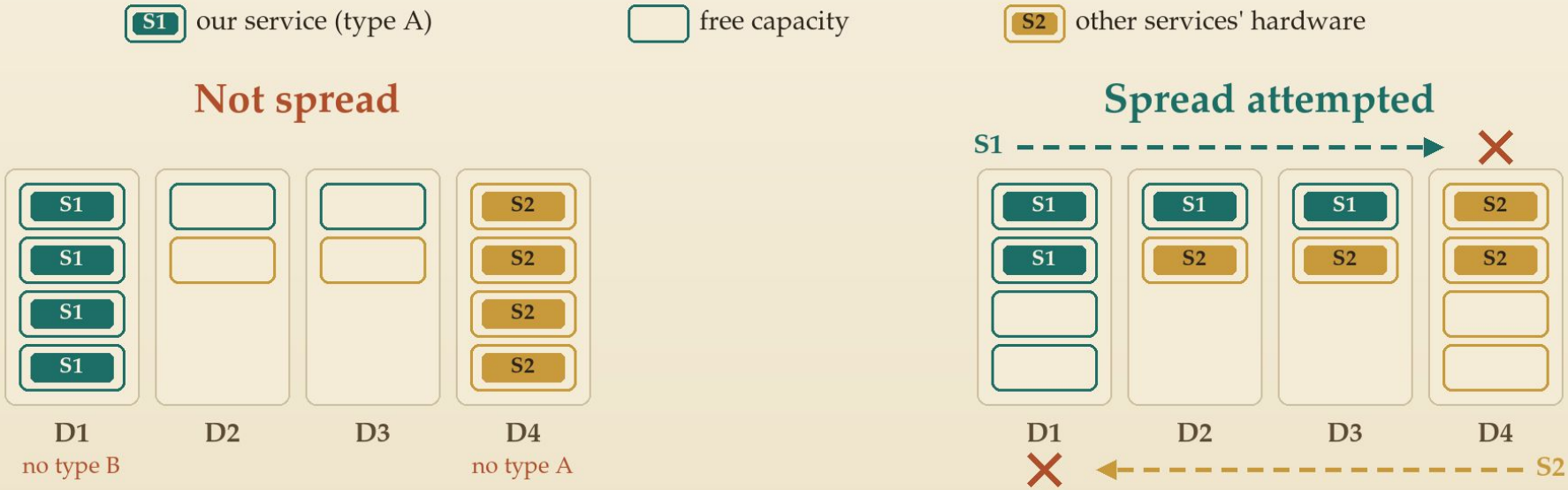
Even spread of h/w & s/w



WHAT

Consistent service expectations

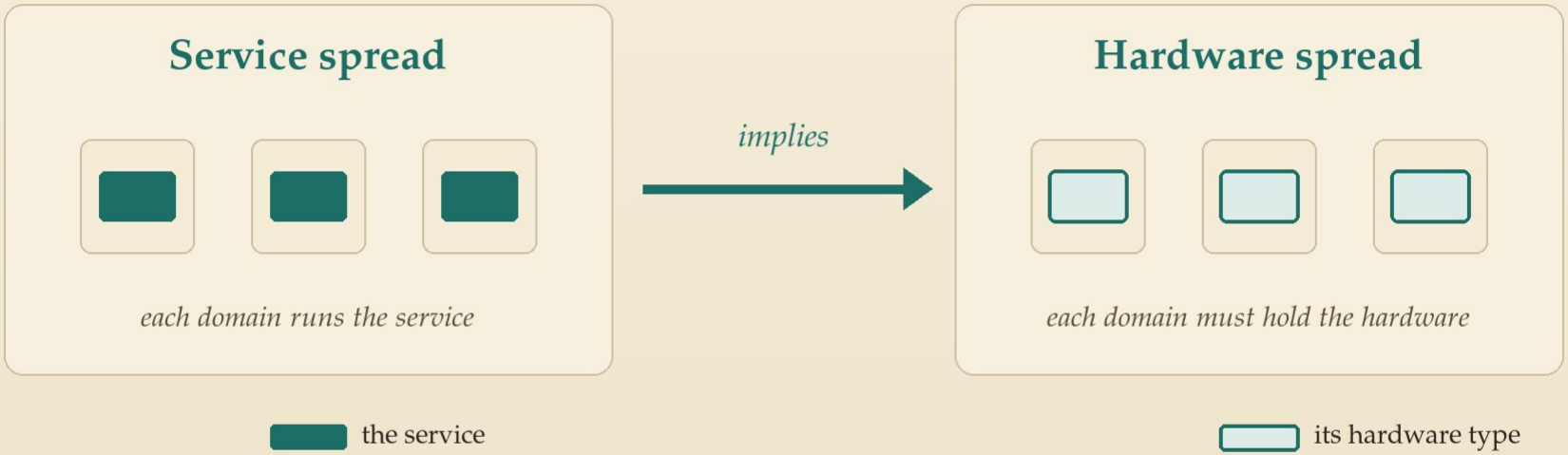
Spreading services across domains



Each service is capped by how its own hardware type is spread

So software spread is bounded by hardware spread

Service spread implies hardware spread

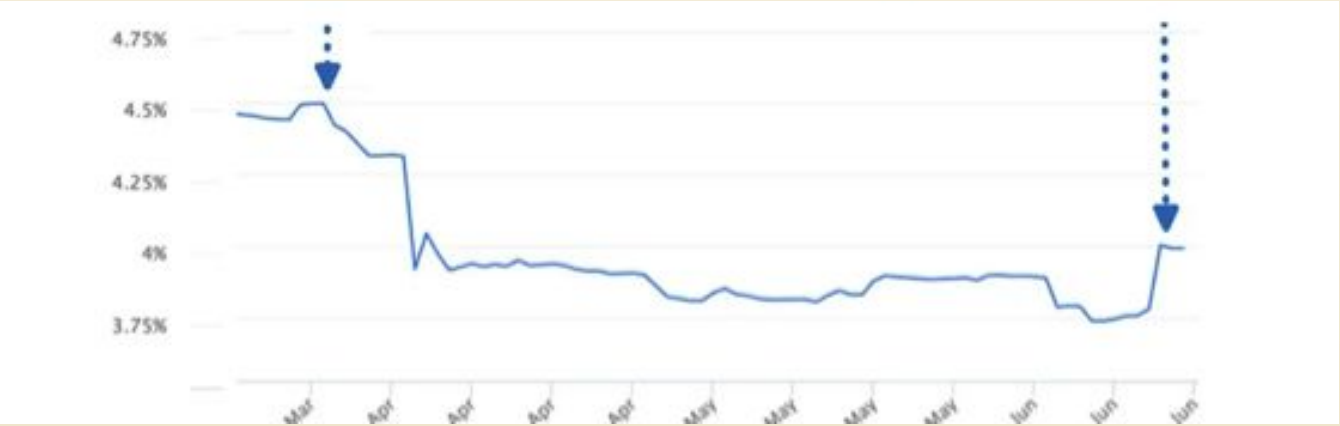


A service reaches only the domains where its hardware lives

Reducing spread in production

~4.5% of fleet

~4% of fleet

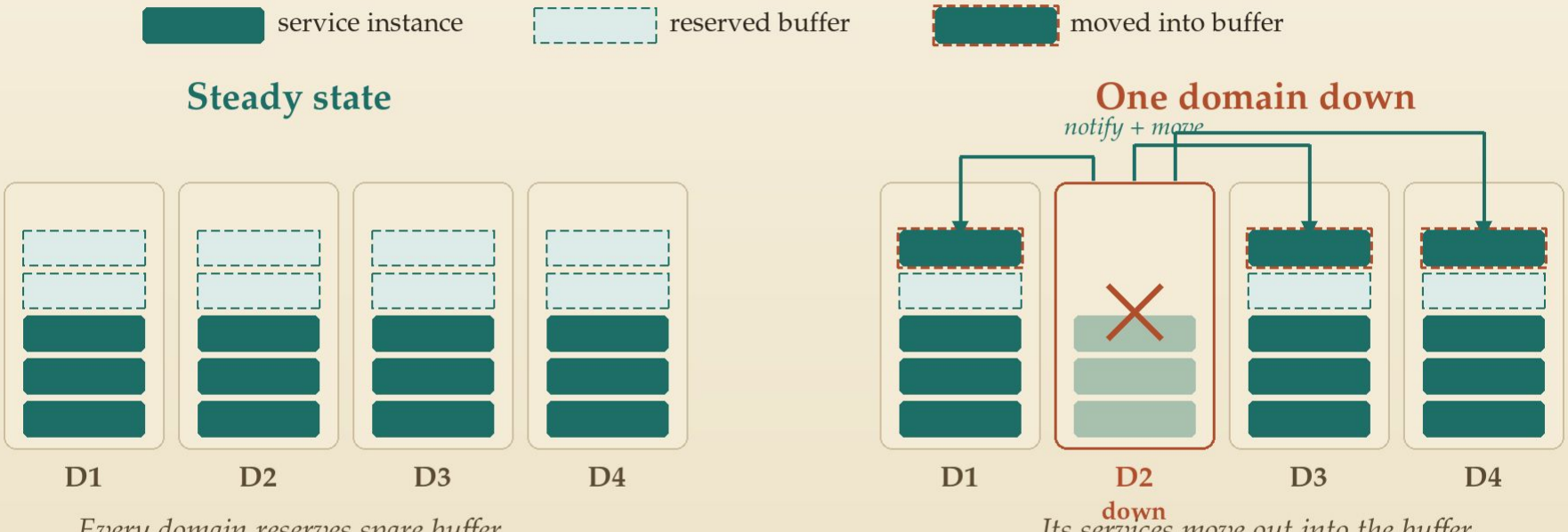


Agenda

1. Context
2. The IAAS contract
3. Why is the contract necessary?
4. How is the contract achieved?
   - 4.1 Spread s/w and h/w
   - **4.2 Multiplexed buffer for 1 domain**
5. What the IAAS contract gives us

Tolerating one domain down

Contract: a service survives any one domain going down



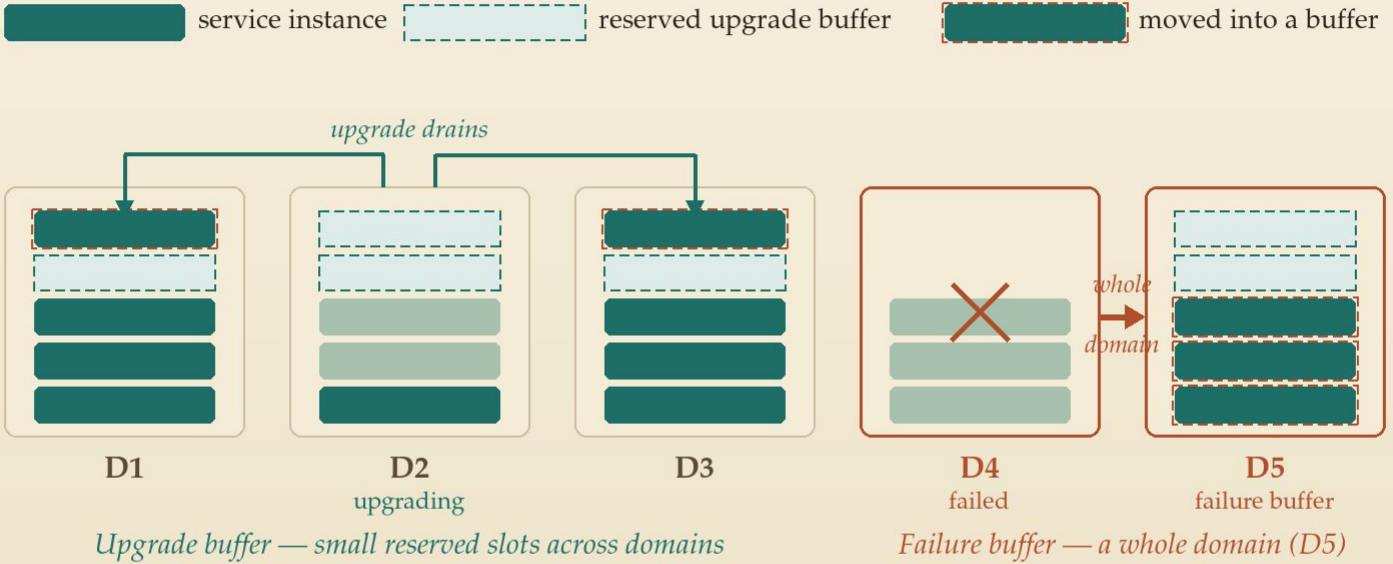
Every domain reserves spare buffer

Its services move out into the buffer

Reserved buffer lets a whole domain drain out — the service stays up

Planning the buffer — Option 1: dedicated

One shared upgrade buffer spread across domains, and a whole failure domain



Upgrade buffer — small reserved slots across domains

Failure buffer — a whole domain (D5)

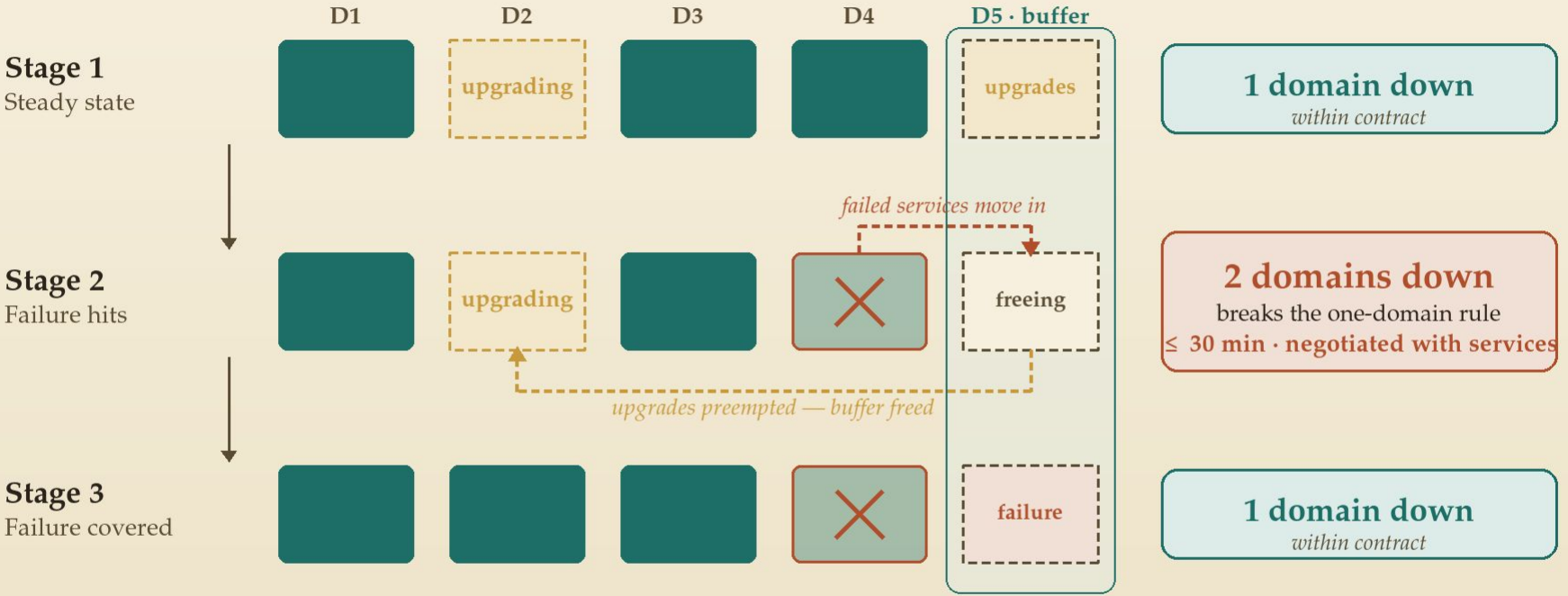
Cost
upgrade buffer + failure buffer (> 1 domain)

Benefit
decoupled, flexible operations

Two dedicated buffers — upgrades and failures never contend, but you pay for both

Planning the buffer — Option 2: multiplexed

One shared buffer, multiplexed — the catch is a brief, bounded window of two domains down



Multiplexing saves a whole buffer — the price is a short two-down window, bounded to 30 minutes

Two ways to plan the buffer — the trade-off

Pay in capacity, or pay in complexity

Trade-off	Option 1 Dedicated buffers	Option 2 Multiplexed buffer (Our choice)
Buffer cost	**> 1 domain** upgrade + failure buffers	**1 domain** one shared buffer
Flexibility	**High** upgrades & failures independent	**Lower** a failure preempts upgrades
Complexity	**Low** simple to operate	**Higher** preemption logic needed

Option 1 buys simplicity with extra capacity; Option 2 buys capacity with extra complexity

One maintenance unit is also the buffer unit

1 maintenance domain = **1 buffer unit**

we maintain and reserve at the same granularity

Agenda

1. Context
2. The IAAS contract
3. Why is the contract necessary?
4. How is the contract achieved?
5. **What the IAAS contract gives us**
   - 5.1 Rollout guarantees
     - 5.1.1 Equivalent domains → uniform drain
     - 5.1.2 Layers of maintenance stack
     - 5.1.3 Train sequence
     - 5.1.4 Rollout results
   - 5.2 DR expectations

The IAAS Contract

WHY

Physical and automated maintenances

HOW

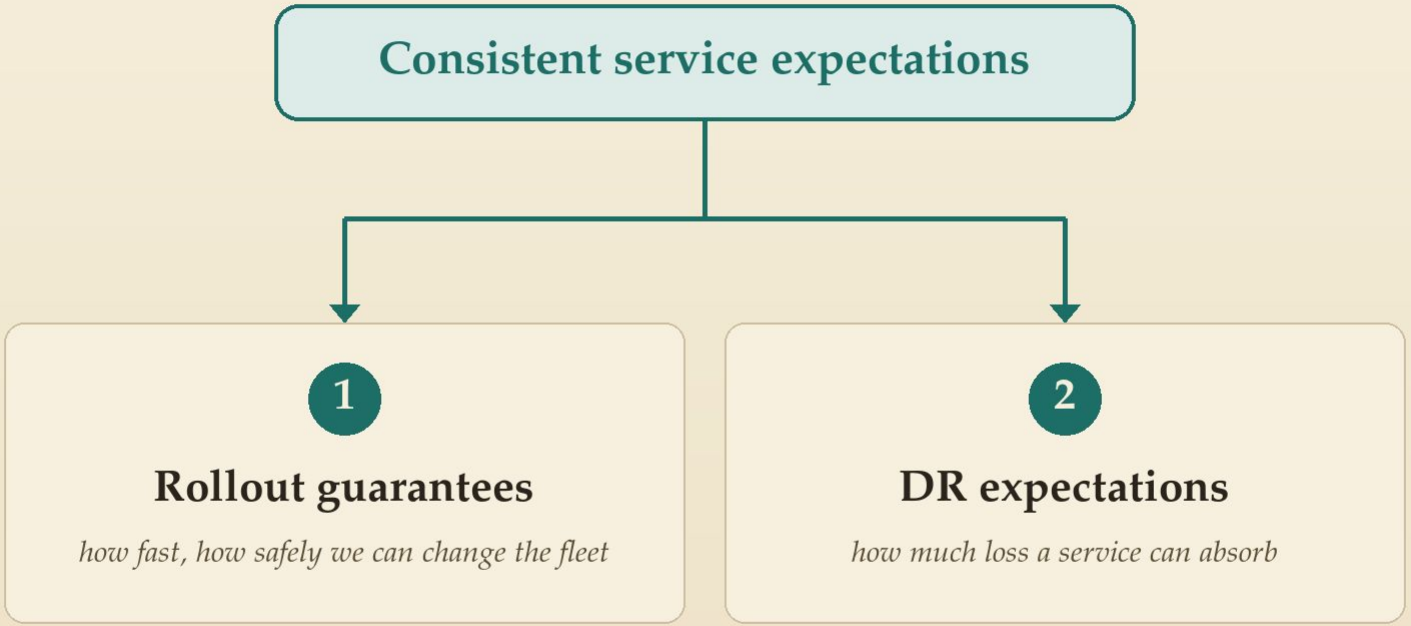
Even spread of h/w & s/w



WHAT

Consistent service expectations

What the IAAS contract gives us

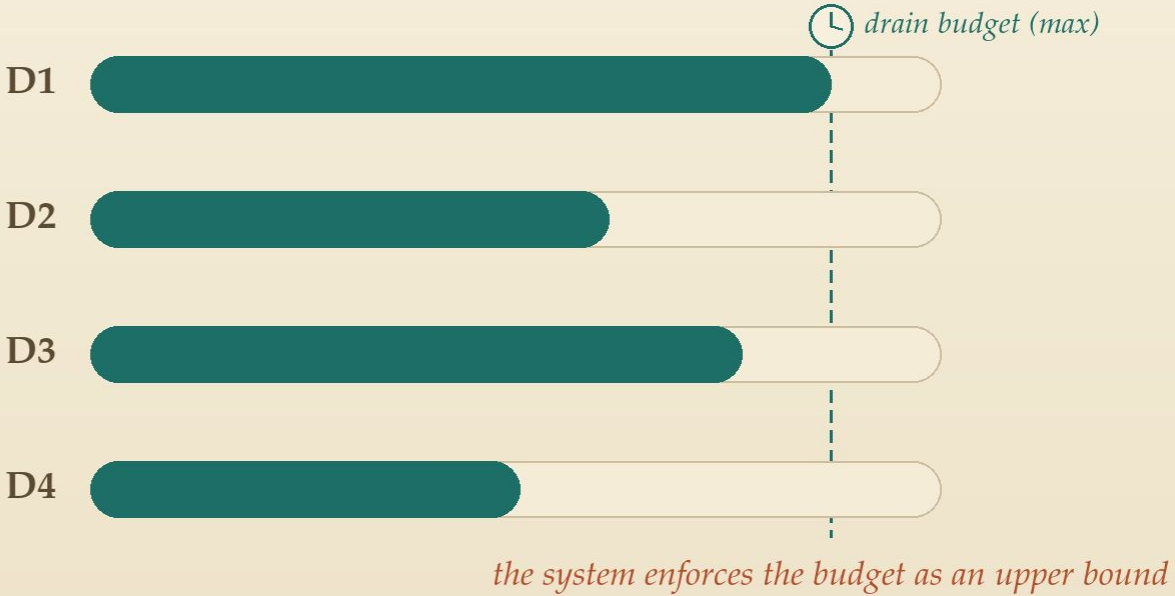


Rollout guarantees

changing the whole fleet, fast and predictably

Equivalent domains drain in equal time

same hardware + same software → equivalent domains



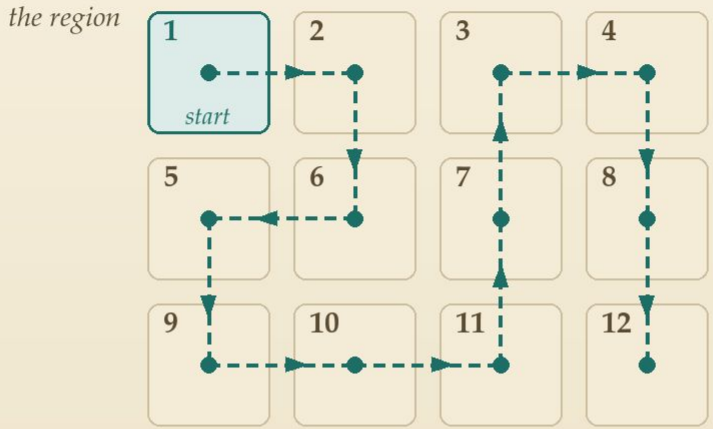
Every domain drains within one budget — many finish sooner

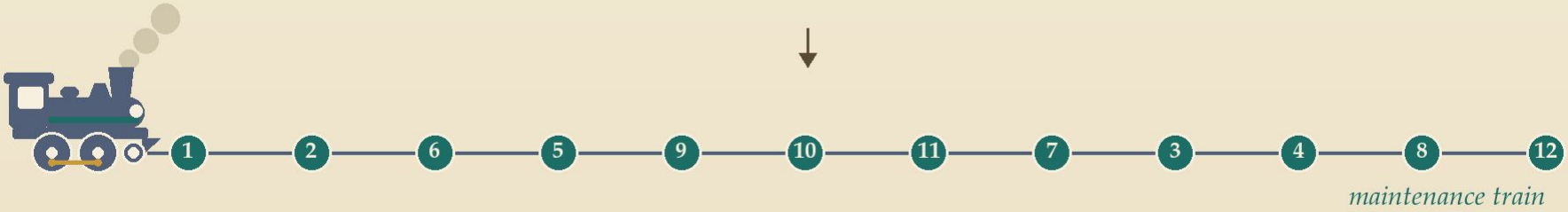
Layered stack — composing SLOs

1	**Intent**	SLO up-to-date intent list · few hours
2	**Sequencing**	SLO visit every MD · 45 days
3	**Budgeting**	SLO tier-aware boxcar plan
4	**Orchestration (Drain)**	SLO drain in 3h · safe execution
5	**Execution**	SLO per-upgrade success · duration SLOs

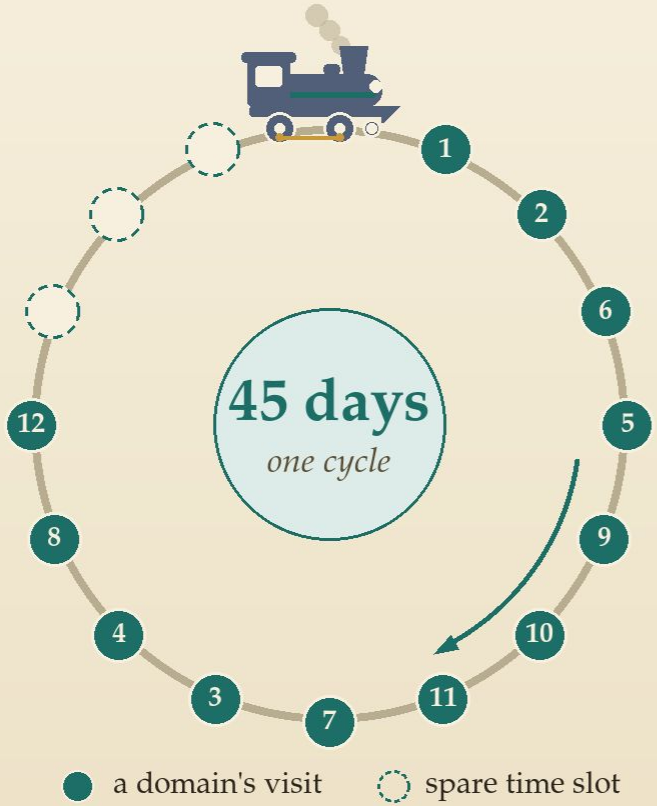
each layer's SLO composes into the next → together they uphold the contract

Covering a region, domain by domain





Folding the train into a cycle



The cycle guarantee

Every domain is visited at least once per cycle

↓

Rollout guarantees

a bound on when any change reaches the whole fleet

Rollout results



rollout success

OS rollout rate

Time for an OS update to reach the whole fleet

years → 45 days

at p99, fleet-wide

what once took years now completes every cycle

Support for physical maintenances



Beyond the contract → Disaster Recovery (DR)

Up to one domain, the buffer absorbs it — past that, DR takes over

the 1-domain threshold

≤ 1 domain down

Sub-region failure

- Within the IaaS contract
- Absorbed by the multiplexed buffer
- Service capacity continues
- Latency stays predictable

> 1 domain down

Region failure

- Beyond the IaaS contract
- Exceeds the multiplexed buffer
- A region-level failure
- DR measures are invoked

Every domain is equivalent — so any single domain's loss fits the buffer.

DR: "Defcon: Preventing Overload with Graceful Feature Degradation" · OSDI '23 · usenix.org/conference/osdi23/presentation/meza

Conclusion

Predictable maintenance at scale

with only a small capacity buffer

Predictable

every domain visited
once per 45-day cycle

At scale

10M+ servers · 10k+ services
one fleet to maintain

Small buffer

one multiplexed domain
covers upgrades and failure

It all rests on one idea — every domain in a region is equivalent

Thank you

Questions?

PIMS · Fleet-wide Datacenter Maintenance at Meta

Amitav Mohanty